life.augmented

UM1766
User manual

Getting started with STM32CubeF3 firmware package for STM32F3 series

Introduction

STMCube™ initiative is an STMicroelectronics original initiative to ease developers life by reducing development efforts, time and cost. STM32Cube covers STM32 portfolio.

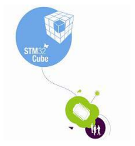
STM32Cube Version 1.x includes:

- The STM32CubeMX, a graphical software configuration tool that allows to generate C initialization code using graphical wizards.
- A comprehensive embedded software platform, delivered per series (such as STM32CubeF3 for STM32F3 series)
  - The STM32Cube HAL, an STM32 abstraction layer embedded software, ensuring maximized portability across STM32 portfolio
  - A consistent set of middleware components such as RTOS, USB, STMTouch, FatFS and Graphics
  - All embedded software utilities coming with a full set of examples

The STMCube™ package is a free solution that can be downloaded from ST website at *http://www.st.com/stm32cube*.

This user manual describes how to get started with the STM32CubeF3 firmware package. *Section 1* describes the main features of STM32CubeF3 firmware, part of the STM32Cube initiative.

Section 2 and *Section 3* provide an overview of the STM32CubeF3 architecture and firmware package structure.



Contents

List of tables

List of figures

1 STM32CubeF3 main features

STM32CubeF3 gathers together, in a single package, all the generic embedded software components required to develop an application on STM32F3 microcontrollers. In line with the STM32Cube^{TM} initiative, this set of components is highly portable, not only within STM32F3 series but also to other STM32 series.

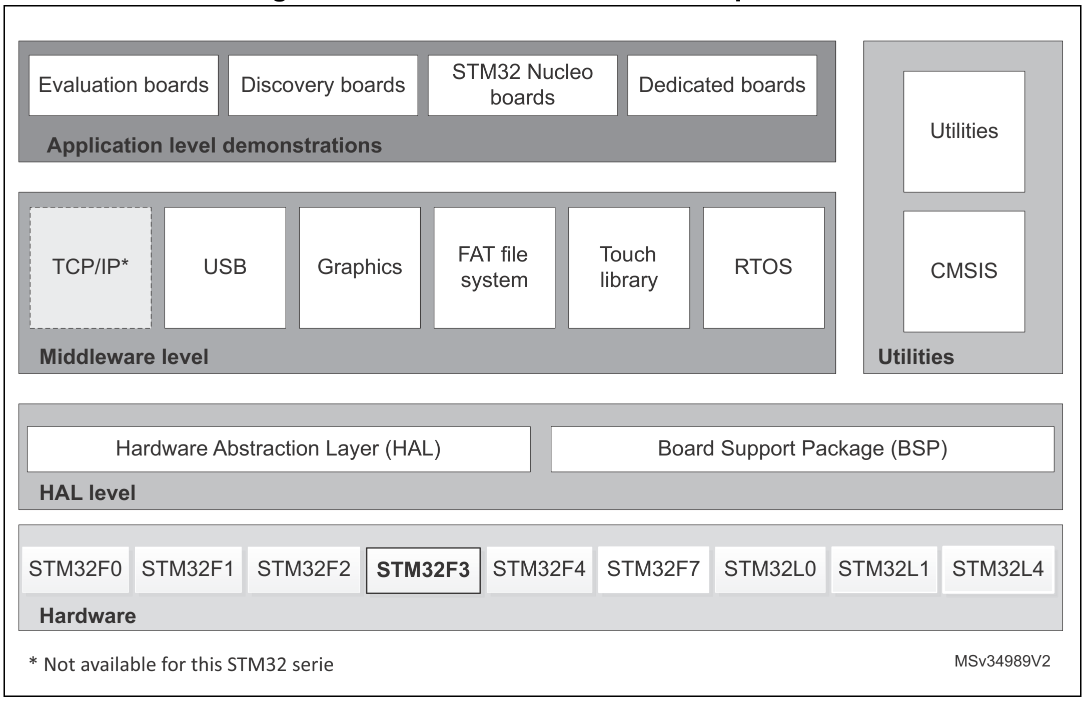
STM32CubeF3 is fully compatible with STM32CubeMX code generator that allows the user to generate initialization code. The package includes a low level hardware abstraction layer (HAL) that covers the microcontroller hardware, together with an extensive set of examples running on STMicroelectronics boards. The HAL is available in open-source BSD license for user convenience.

STM32CubeF3 package also contains a set of middleware components with the corresponding examples. They come in very permissive license terms:

- Full USB Device stack supporting many classes: Audio, HID, MSC, CDC and DFU,
- CMSIS-RTOS implementation with FreeRTOS open source solution,
- FAT File system based on open source FatFS solution,
- STMTouch touch sensing library solution,
- STemWin, a professional graphical stack solution available in binary format and based on ST partner solution SEGGER emWin.

Several applications and demonstrations implementing all these middleware components are also provided in the STM32CubeF3 package.

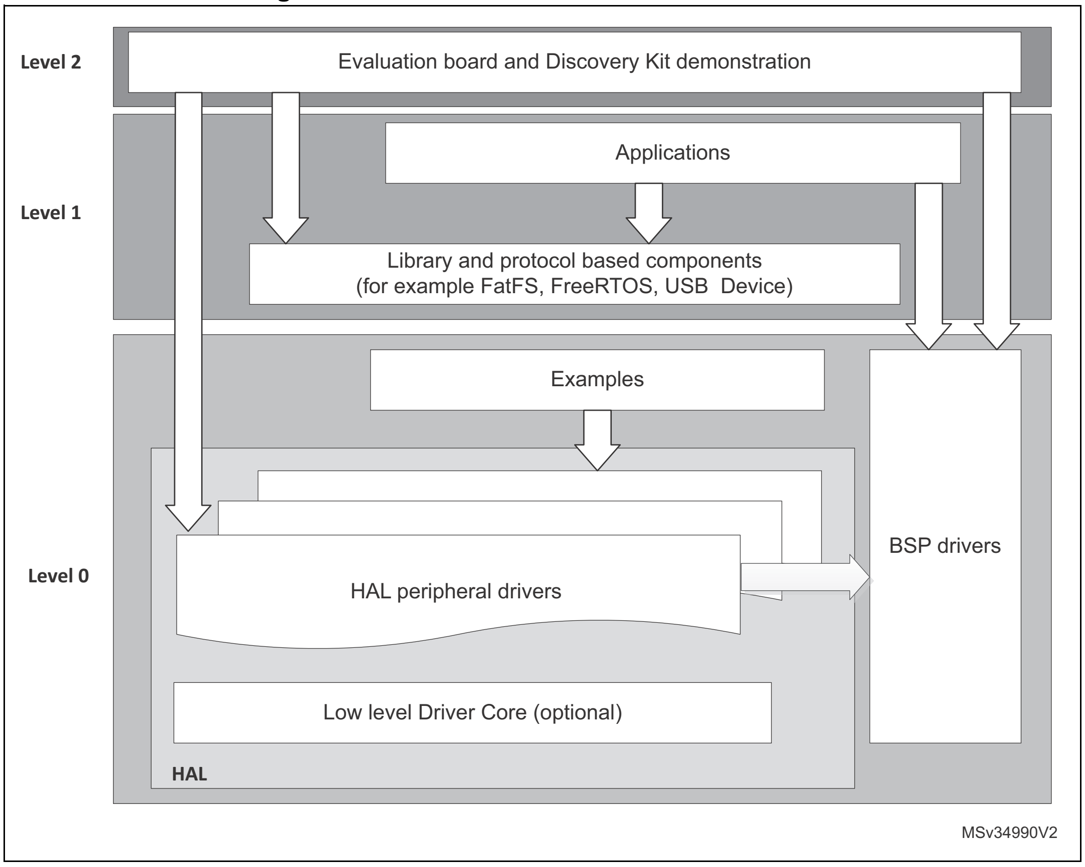
Figure 1. STM32Cube^{TM} firmware components



2 STM32CubeF3 architecture overview

The STM32Cube firmware solution is built around three independent levels that can easily interact with each other's as described in the *Figure 2* below:

Figure 2. STM32CubeF3 firmware architecture


Level 2
Evaluation board and Discovery Kit demonstration
Level 1
Applications
Library and protocol based components
(for example FatFS, FreeRTOS, USB Device)
Level 0
Examples
BSP drivers
HAL peripheral drivers
Low level Driver Core (optional)
HAL
MSv34990V2

Level 0: This level is divided into three sub-layers:

- **Board Support Package (BSP)**: this layer offers a set of APIs relative to the hardware components in the hardware boards (for example LCD drivers, and microSD) and composed of two parts:
  - **Component**: driver relative to the external device on the board and not related to the STM32. The component driver provides specific APIs to the BSP driver external components and can be portable on any other board.
  - **BSP driver**: it permits to link the component driver to a specific board and provides a set of friendly used APIs. The APIs naming rule is BSP_FUNCT_Action(): ex. BSP_LED_Init(),BSP_LED_On()

  BSP is based on a modular architecture allowing an easy porting on any hardware by implementing the low level routines.
- Hardware Abstraction Layer (HAL): this layer provides the low level drivers and the hardware interfacing methods to interact with the upper layers (application, libraries and stacks). It provides generic, multi instance and functionalities oriented APIs which permit to offload the user application implementation by providing ready to use

process. As example, for the communication peripherals (for example I2S, UART) it provides APIs allowing to initialize and configure the peripheral, manage data transfer based on polling, interrupt or DMA process, and manage communication errors that may raise during communication. The HAL Drivers APIs are split into two categories: the generic API category which provides common and generic functions to all the STM32 series and the extension API category which provides specific and customized functions for a specific family or a specific part number.

- Basic peripheral usage examples: this layer encloses the examples built over the STM32 peripheral using only the HAL and BSP resources.

Level 1: this level is divided into two sub-layers:

- **Middleware components**: set of Libraries covering USB Device library, STMTouch touch sensing library, graphical STemWin library, FreeRTOS and FatFS. Horizontal interactions between the components of this layer is done directly by calling the feature APIs while the vertical interaction with the low level drivers is done through specific callbacks and static macros implemented in the library system call interface. As example, the FatFs implements the disk I/O driver to access microSD drive or the USB Mass Storage Class.

  The main features of each Middleware component are as follows:

  USB Device Library
  - Several USB classes supported (Mass-Storage, HID, CDC, DFU, AUDIO, MTP)
  - Supports multi packet transfer features: allows sending big amounts of data without splitting them into max packet size transfers.
  - Uses configuration files to change the core and the library configuration without changing the library code (Read Only).
  - RTOS and Standalone operation,
  - The link with low-level driver is done through an abstraction layer using the configuration file to avoid any dependency between the Library and the low-level drivers.

  FreeRTOS
  - Open source standard,
  - CMSIS compatibility layer,
  - Tickless operation during low-power mode,
  - Integration with all STM32Cube Middleware modules.

  FAT File system
  - FATFS FAT open source library,
  - Long file name support,
  - Dynamic multi-drive support,
  - RTOS and standalone operation,
  - Examples with microSD.

  STM32 Touch Sensing Library
  - Robust STMTouch capacitive touch sensing solution supporting proximity, touchkey, linear and rotary touch sensor using a proven surface charge transfer acquisition principle.

  STemWin Library
  - Graphical library supporting LCD provided as part as the STM32CubeF3 firmware package.

- **Examples based on the Middleware components**: each Middleware component comes with one or more examples (called also Applications) showing how to use it. Integration examples that use several Middleware components are also provided.

Level 2: this level is composed of a single layer which is a global real-time and graphical demonstration based on the Middleware service layer, the low level abstraction layer and the basic peripheral usage applications for board based functionalities.

3 STM32CubeF3 firmware package overview

3.1 Supported STM32F3 devices and hardware

STM32Cube^{TM} offers highly portable Hardware Abstraction Layer (HAL) built around a generic architecture and allows the build-upon layers, like the middleware layer, to implement its functions without knowing, in-depth, the MCU used. This improves the library code re-usability and guarantees an easy portability on other devices.

The layered architecture of the STM32CubeF3 offers the full support of the whole STM32F3 series. The user only needs define the right macro in *stm32f3xx.h*.

Table 1 below provides the macro to define depending on the used STM32F3 device (this macro must also be defined in the compiler preprocessor).

Table 1. Macros for STM32F3 series

Macro defined in stm32f3xx.h	STM32F3 devices
STM32F301x8	STM32F301K6, STM32F301C6, STM32F301R6, STM32F301K8, STM32F301C8 and STM32F301R8
STM32F302x8	STM32F302K6, STM32F302C6, STM32F302R6, STM32F302K8, STM32F302C8 and STM32F302R8
STM32F302xC	STM32F302CB, STM32F302RB, STM32F302VB, STM32F302CC, STM32F302RC and STM32F302VC
STM32F302xE	STM32F302RD, STM32F302VD, STM32F302ZD, STM32F302RE, STM32F302VE and STM32F302ZE
STM32F303x8	STM32F303K6, STM32F303C6, STM32F303R6, STM32F303K8, STM32F303C8 and STM32F303R8
STM32F303xC	STM32F303CB, STM32F303RB, STM32F303VB, STM32F303CC, STM32F303RC and STM32F303VC
STM32F303xE	STM32F303RD, STM32F303VD, STM32F303ZD, STM32F303RE, STM32F303VE and STM32F303ZE
STM32F373xC	STM32F373C8, STM32F373R8, STM32F373V8, STM32F373CB, STM32F373RB, STM32F373VB, STM32F373CC, STM32F373RC and STM32F373VC
STM32F334x8	STM32F334K4, STM32F334C4, STM32F334R4, STM32F334K6, STM32F334C6, STM32F334R6, STM32F334K8, STM32F334C8 and STM32F334R8
STM32F318xx	STM32F318K8 and STM32F318C8
STM32F328xx	STM32F328C8 and STM32F328R8
STM32F358xx	STM32F358CC, STM32F358RC and STM32F358VC
STM32F378xx	STM32F378CC, STM32F378RC and STM32F378VC
STM32F398xx	STM32F398RE, STM32F398VE and STM32F398ZE

STM32CubeF3 features a rich set of examples and applications at all levels making it easy to understand and use any HAL driver and/or Middleware components. These examples are running on STMicroelectronics boards as listed in *Table 2* below:

Table 2. Boards for STM32F3 series

Board part number	STM32F3 devices supported
NUCLEO-F303RE	STM32F303RE
STM32303E-EVAL	STM32F303VE
32F3348DISCOVERY	STM32F334C8
NUCLEO-F334R8	STM32F334R8
NUCLEO-F302R8	STM32F302R8
STM32373C-EVAL	STM32F373VC
NUCLEO-F303K8	STM32F303K8
NUCLEO-F303ZE	STM32F303ZE
STM32F3DISCOVERY	STM32F303VC
STM32303C-EVAL	STM32F303VC

STM32CubeF3 support both Nucleo-32, Nucleo-64 and Nucleo-144 boards.

- Nucleo-64 and Nucleo-144 boards support Adafruit LCD display Arduino™ UNO shields which embed a microSD connector and a joystick in addition to the LCD.
- Nucleo-32 boards support Gravitech 7-segment display Arduino™ NANO shields which allow displaying up to four-digit numbers and characters.

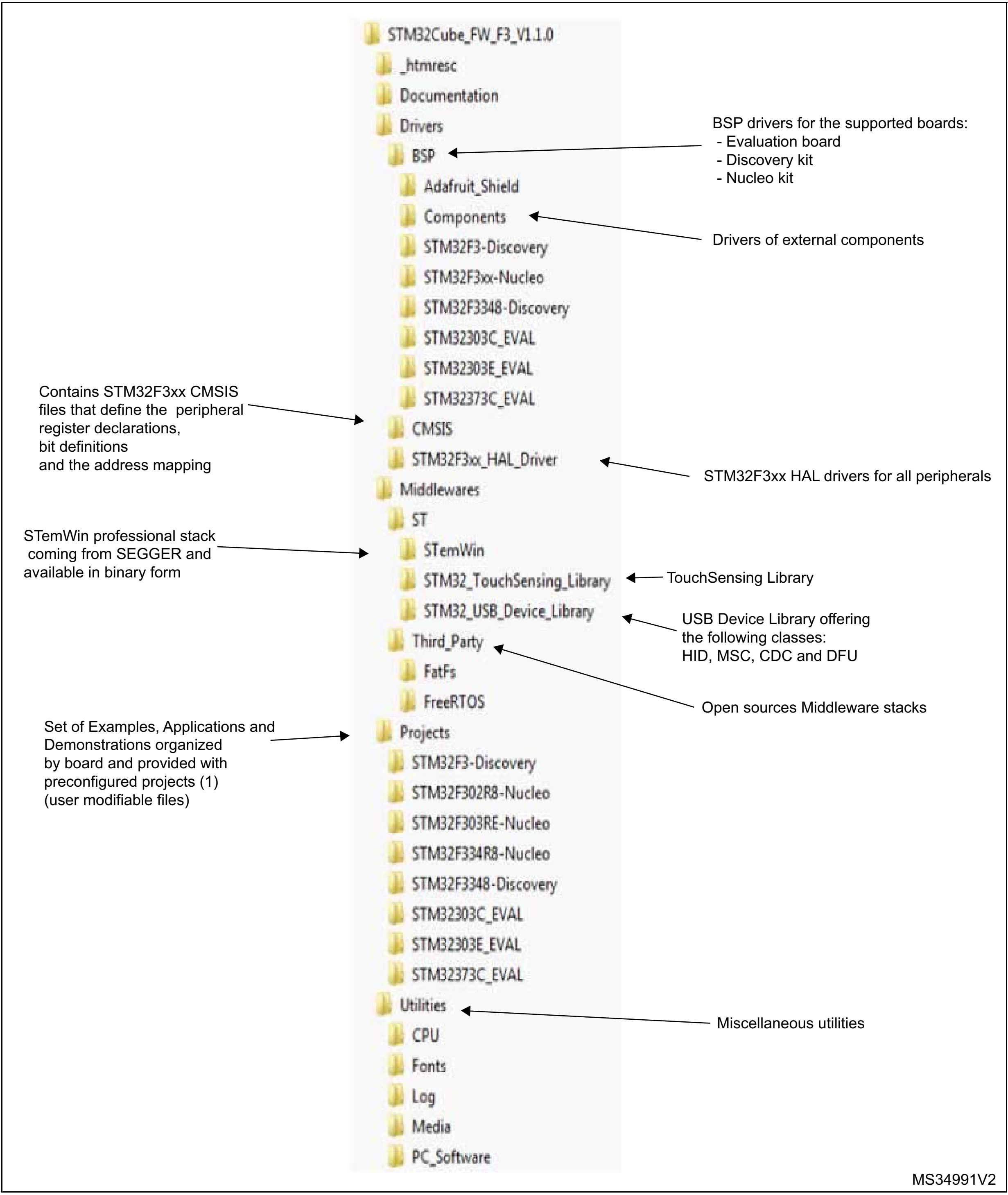
The Arduino™ shield drivers are provided within the BSP component. Their usage is illustrated by a demonstration firmware.

The STM32CubeF3 firmware can run on any compatible hardware. Simply update the BSP drivers to port the provided examples on your board if its hardware features are the same (e.g. LED, LCD display, buttons).

3.2 Firmware package overview

The STM32CubeF3 firmware solution is provided in one single zip package having the structure shown in *Figure 3* hereafter.

Figure 3. STM32CubeF3 firmware package structure

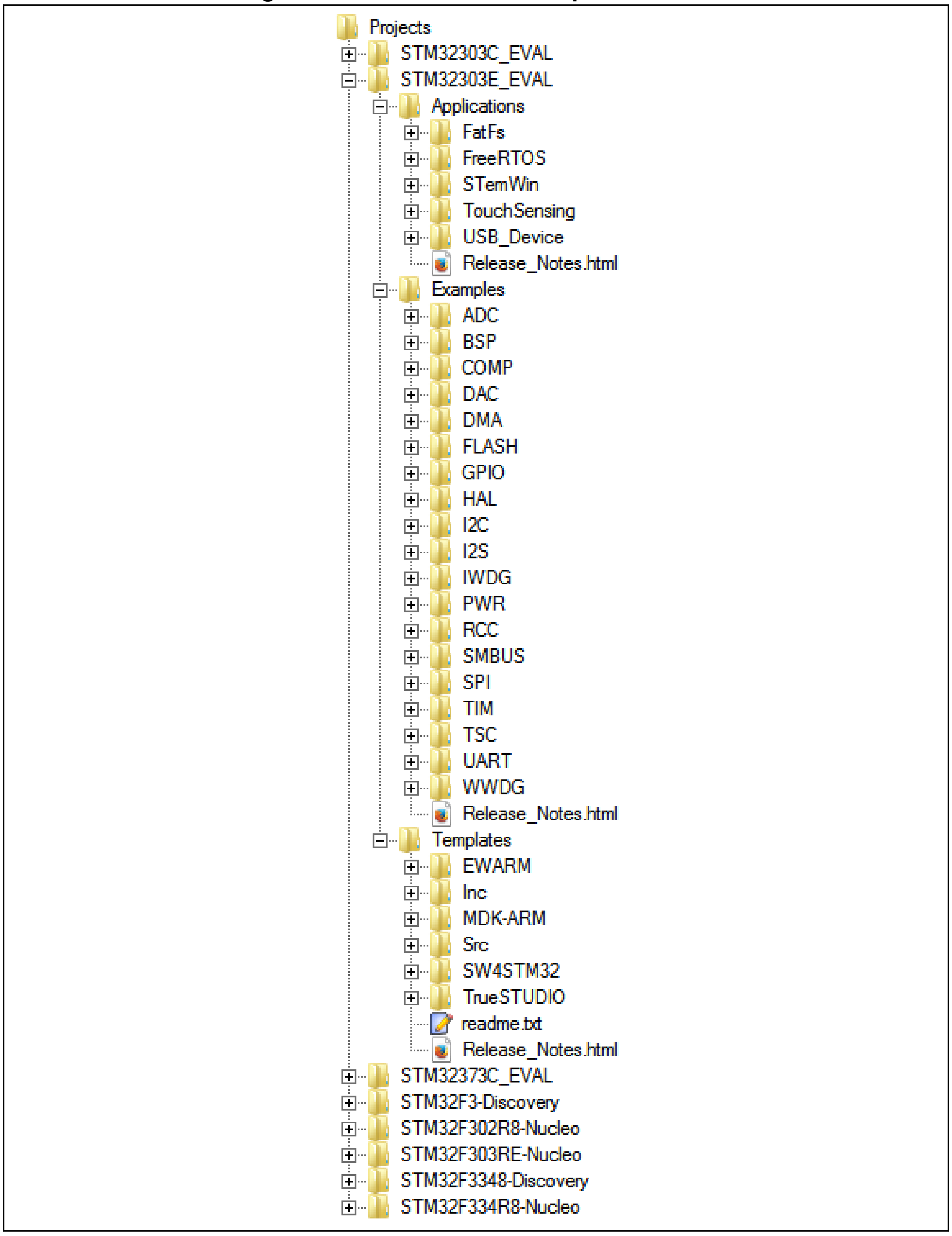

STM32Cube_FW_F3_V1.1.0
_htmresc
Documentation
Drivers
BSP
Adafruit_Shield
Components
STM32F3-Discovery
STM32F3xx-Nucleo
STM32F3348-Discovery
STM32303C_EVAL
STM32303E_EVAL
STM32373C_EVAL
CMSIS
STM32F3xx_HAL_Driver
Middlewares
ST
STemWin
STM32_TouchSensing_Library
STM32_USB_Device_Library
Third_Party
FatFs
FreeRTOS
Projects
STM32F3-Discovery
STM32F302R8-Nucleo
STM32F303RE-Nucleo
STM32F334R8-Nucleo
STM32F3348-Discovery
STM32303C_EVAL
STM32303E_EVAL
STM32373C_EVAL
Utilities
CPU
Fonts
Log
Media
PC_Software
BSP drivers for the supported boards: - Evaluation board - Discovery kit - Nucleo kit
Drivers of external components
Contains STM32F3xx CMSIS files that define the peripheral register declarations, bit definitions and the address mapping
STM32F3xx HAL drivers for all peripherals
STemWin professional stack coming from SEGGER and available in binary form
TouchSensing Library
USB Device Library offering the following classes: HID, MSC, CDC and DFU
Open sources Middleware stacks
Set of Examples, Applications and Demonstrations organized by board and provided with preconfigured projects (1) (user modifiable files)
Miscellaneous utilities
MS34991V2

1. The set of examples, applications and demonstrations stored in the Projects folder are user modifiable files.

For each board, a set of examples are provided with pre-configured projects for EWARM, MDK-ARM, TrueSTUDIO and SW4STM32 toolchains.

Figure 4 shows the projects structure for the STM32303E-EVAL board.

Figure 4. STM32CubeF3 example overview



```
Projects
  STM32303C_EVAL
  STM32303E_EVAL
    Applications
      FatFs
      FreeRTOS
      STemWin
      TouchSensing
      USB_Device
      Release_Notes.html
    Examples
      ADC
      BSP
      COMP
      DAC
      DMA
      FLASH
      GPIO
      HAL
      I2C
      I2S
      IWDG
      PWR
      RCC
      SMBUS
      SPI
      TIM
      TSC
      UART
      WWDG
      Release_Notes.html
    Templates
      EWARM
      Inc
      MDK-ARM
      Src
      SW4STM32
      TrueSTUDIO
      readme.txt
      Release_Notes.html
  STM32373C_EVAL
  STM32F3-Discovery
  STM32F302R8-Nucleo
  STM32F303RE-Nucleo
  STM32F3348-Discovery
  STM32F334R8-Nucleo
```

The examples are organized as per the STM32Cube level they apply to, and are named as described below:

- Examples in level 0 are called *Examples*. They use the HAL drivers without any Middleware component.
- Examples in level 1 are called *Applications*. They provide typical use cases of each Middleware component.

The *Template* project available in the Template directory allows to quickly build any firmware application on a given board.

All examples have the same structure,

- *\Inc* folder that contains all header files
- *\Src* folder for the sources code
- *\EWARM, \MDK-ARM*, *\TrueSTUDIO* and *\SW4STM32* folders contain the pre-configured project for each toolchain.
- *readme.txt* describing the example behavior and needed environment to make it working

Table 3 provides the number of projects available for each board.

Table 3. Number of examples available for each board

Board	Templates	Examples	Demonstration	Applications	Total
NUCLEO-F303RE	1	27	1	8	**37**
STM32303E-EVAL	1	42	0	18	**61**
32F3348DISCOVERY	1	22	1	1	**25**
NUCLEO-F334R8	1	2	1	1	**5**
NUCLEO-F302R8	1	29	1	3	**34**
STM32373C-EVAL	1	40	0	21	**62**
NUCLEO-F303K8	1	27	1	1	**30**
NUCLEO-F303ZE	1	39	1	3	**44**
STM32F3DISCOVERY	1	38	1	3	**43**
STM32303C-EVAL	1	54	0	15	**70**
Total	**10**	**320**	**7**	**74**	**411**

4 Getting started

4.1 Running your first example

This section explains how to run a first example within STM32CubeF3, using as illustration the generation of a simple LED toggle running on STM32F302R8 Nucleo board:

1. **Download** the STM32CubeF3 firmware package. **Unzip** it into a directory of your choice. Make sure not to modify the package structure shown in *Figure 4*. Note that it is also recommended to copy the package at a location close to your root volume (for example `C:\Eval` or `G:\Tests`) because some IDEs encounter problems when the path length is too long.
2. **Browse** to `\Projects\STM32F302R8-Nucleo\Examples`
3. **Open** *\GPIO*, then *\GPIO_EXTI* folder
4. Open the project with your preferred toolchain (*)
5. Rebuild all files and load your image into target memory
6. Run the example: each time you press the USER push button, the LED2 toggles (for more details, refer to the example *readme file*).

(*) The following section provides a quick overview on how to open, build and run an example with the supported toolchains:

- EWARM
  - Under the example folder, **open** \EWARM subfolder
  - Launch the *Project.eww* workspace (**)
  - **Rebuild all files**: Project->Rebuild all
  - **Load the project image**: Project->Debug
  - **Run the program**: Debug->Go(F5)
- MDK-ARM
  - Under the example folder, **open** \MDK-ARM subfolder
  - **Launch** the Project.uvproj workspace (**)
  - **Rebuild all the files:** Project->Rebuild all target files
  - **Load the project image**: Debug->Start/Stop Debug Session
  - **Run the program**: Debug->Run (F5)
- TrueSTUDIO
  - **Open** the TrueSTUDIO toolchain
  - **Select** *File->Switch Workspace->Other* and **browse** to TrueSTUDIO workspace directory
  - **Select** *File->Import*, **select** *General->'Existing Projects into Workspace*' and then **Select** "Next".
  - **Browse** to the *TrueSTUDIO* workspace directory, select the project
  - **Rebuild all the project files**: select the project in the "*Project explorer"* window then **select** *Project->build project* menu.
  - Run the program: Run->Debug (F11)
- SW4STM32
  a) Open the SW4STM32 toolchain.
  b) Click **File->Switch Workspace->Other** and browse to the SW4STM32 workspace directory.
  c) Click **File->Import**, select **General->'Existing Projects into Workspace'** and then click "**Next**".
  d) Browse to the SW4STM32 workspace directory and select the project.
  e) Rebuild all project files: select the project in the "**Project explorer**" window then click **Project->build project** menu.
  f) Run program: **Run->Debug** (F11).

(**) The workspace name may change from one example to another.

4.2 Developing your own application

This section describes the steps required to create your own application using STM32CubeF3.

1. **Create your project**: to create a new project you can either start from the Template project provided for each board under `\Projects\<STM32xxx_yyy>\Templates` or from any available project under `\Projects\<STM32xxy_yyy>\Examples` or

`\Projects\<STM32xx_yyy>\Applications` (where <STM32xxx_yyy> refers to the board name, for example STM32303C_EVAL).

The Template project provides an empty main loop function. It is a good starting point to get familiar with project settings for the STM32CubeF3. The template has the following characteristics:

a) It contains sources of HAL, CMSIS and BSP drivers which are the minimal components to develop a code on a given board.

b) It contains the include paths for all the firmware components.

c) It defines the STM32F3 device supported, allowing to configure the CMSIS and HAL drivers accordingly.

d) It provides ready-to-use user files pre-configured as shown below

- HAL is initialized with the default time base with ARM Core SysTick,
- SysTick ISR is implemented for HAL_Delay() purpose,
- System clock is configured with the minimum frequency of the device (HSI) for an optimum power consumption.

Note: when copying an existing project to another location, make sure to update the include paths.

2. **Add the necessary Middleware to your project (optional)**: the available Middleware stacks are: USB Device Library, STemWin, Touch Sensing Library, FreeRTOS and FatFS. To know which source files you need to add in the project files list, refer to the documentation provided for each Middleware. You may also look at the Applications available under `\Projects\STM32xxx_yyy\Applications\<MW_Stack>` (where <MW_Stack> refers to the Middleware stack, for example USB_Device) to see which sources files and include paths need to be added.
3. **Configure the firmware components**: the HAL and Middleware components offer a set of build time configuration options using macros “#define” declared in a header file. A template configuration file is provided within each component, it has to be copied to the project folder (usually the configuration file is named *xxx_conf_template.h*, the word “_template” needs to be removed when copying the file into the project folder). The configuration file provides enough information to know the impact of each configuration option; more detailed information is available in the documentation provided for each component.
4. **Start the HAL Library**: after jumping to the main program, the application code calls the *HAL_Init()* API to initialize the HAL Library, which does the following:

   a) configuration of the Flash prefetch and SysTick interrupt priority (configured by user through macros defined in *stm32f3xx_hal_conf.h*),

   b) configuration of the SysTick to generate an interrupt each 1 msec at the SysTick interrupt priority TICK_INT_PRIO defined in *stm32f3xx_hal_conf.h*, which is clocked by the HSI (at this stage, the clock is not yet configured and thus the system is running from the internal HSI at 8 MHz),

   c) Setting of NVIC Group Priority to 4,

   d) Calling of HAL_MspInit() callback function defined in the user file *stm32f3xx_hal_msp.c,* to run the global low level hardware initializations.
5. **Configure the system clock:** the system clock configuration is done by calling the two APIs described below:

   a) HAL_RCC_OscConfig(): configures the internal and/or external oscillators, PLL source and factors. The user may select to configure one oscillator or all oscillators. The PLL configuration can be skipped if there is no need to run the system at high frequency.

b) HAL_RCC_ClockConfig(): configures the system clock source, the Flash latency and AHB and APB prescalers.

The parameters of the clock configuration functions can be evaluated thanks to the Clock Configuration tab of the STM32CubeMX tool.

6. Peripheral initialization
   a) Start by writing the peripheral HAL_PPP_MspInit function. For this function, please proceed as follows:
      - Enable the peripheral clock.
      - Configure the peripheral GPIOs.
      - Configure the DMA channel and enable the DMA interrupt (if needed).
      - Enable the peripheral interrupt (if needed).
   b) Edit the *stm32xxx_it.c* to call the required interrupt handlers (peripheral and DMA), if needed.
   c) Write process complete callback functions if you plan to use peripheral interrupt or DMA.
   d) In your *main.c* file, initialize the peripheral handle structure then call the function HAL_PPP_Init() to initialize your peripheral.
7. **Developing your application process:** at this stage, your system is ready and you can start developing your application code.
   a) The HAL provides intuitive and ready to use APIs to configure the peripheral, and support polling, IT and DMA programming model, to accommodate any application requirements. For more details on how to use each peripheral, refer to the extensive set of examples provided.
   b) If your application has some real time constraints, you can find a large set of examples showing how to use FreeRTOS and its integration with all Middleware stacks provided within STM32CubeF3. This can be a good starting point for your development.

Caution: In the default HAL implementation, SysTick timer is the source of time base. It is used to generate interrupts at regular time intervals. Take care if HAL_Delay() is called from the peripheral ISR process. The SysTick interrupt must have higher priority (numerically lower) than the peripheral interrupt. Otherwise, the caller ISR process is blocked. Functions affecting the time base configurations are declared as __weak to make the override possible in case of other implementations in user file (using a general purpose timer for example or other time source). For more details please refer to HAL_TimeBase example.

4.3 Using STM32CubeMX to generate the initialization C code

An alternative to steps 1 to 6 described in *Section 4.2* consists of using the STM32CubeMX tool to generate the code for the initialization of the system, the peripherals and middleware (steps 1 to 6 above) through a step-by-step process:

- Select the STMicroelectronics STM32 microcontroller that matches the required set of peripherals.
- Configure each required embedded software using the pinout-conflict solver, a clock-tree setting helper, a power consumption calculator, and the utility performing MCU peripheral configuration (for example GPIO, USART) and middleware stacks (for example USB).
- Generate the initialization C code based on the configuration selected. This code is ready to use within several development environments. The user code is kept at the next code generation.

For more information, please refer to UM1718.

4.4 Getting STM32CubeF3 release updates

The STM32CubeF3 firmware package comes with an updater utility: STM32CubeUpdater, also available as a menu within STM32CubeMX code generation tool.

The updater solution detects new firmware releases and patches available on *www.st.com* and proposes the download on the user's computer.

Installing and running the STM32CubeUpdater program

The *STM32CubeUpdater.exe* is available under `\Utilities\PC_Software`.

- Double-click the *SetupSTM32CubeUpdater.exe* file to launch the installation.
- Accept the license terms and follow the different installation steps.

Upon successful installation, STM32CubeUpdater becomes available as an STMicroelectronics program under Program Files and is automatically launched. The STM32CubeUpdater icon appears in the system tray:

Right-click the updater icon and select **Updater Settings** to configure the Updater connection and whether to perform manual or automatic checks (see STM32CubeMX User guide - UM1718 section 3 - for more details on Updater configuration).

5 FAQs

5.1 What is the license scheme for the STM32CubeF3 firmware?

The HAL is distributed under a non-restrictive BSD (Berkeley Software Distribution) license. The Middleware stacks made by ST (USB Host and Device Libraries, STemWin) come with a licensing model allowing easy reuse, provided it runs on an ST device.

The Middleware based on well-known open-source solutions (FreeRTOS and FatFs) have user-friendly license terms. For more details, refer to the license agreement of each Middleware.

5.2 Which boards are supported by the STM32CubeF3 firmware package?

The STM32CubeF3 firmware package provides BSP drivers and ready-to-use examples for the following STM32F3 boards: STM32303C-EVAL, STM32303E-EVAL, STM32373C-EVAL, STM32F3DISCOVERY, 32F3348DISCOVERY, NUCLEO-F302R8, NUCLEO-F303RE, NUCLEO-F303ZE, NUCLEO-F334R8 and NUCLEO-F303K8..

5.3 Is there any link with Standard Peripheral Libraries?

The STM32Cube HAL Layer is the replacement of the Standard Peripheral Library.

The HAL APIs offer a higher abstraction level compared to the standard peripheral APIs. HAL focuses on peripheral common functionalities rather than hardware. The higher abstraction level allows to define a set of user friendly APIs that can be easily ported from one product to another.

Although the existing Standard Peripheral Libraries are supported, they are not recommended for new designs.

5.4 Does the HAL take benefit from interrupts or DMA? How can this be controlled?

Yes. The HAL supports three API programming models: polling, interrupt and DMA (with or without interrupt generation).

5.5 Are any examples provided with the ready-to-use toolset projects?

Yes. STM32CubeF3 provides an extensive set of examples and applications (around 70 for STM32303C-EVAL). They come with the pre-configured project of several tool sets: IAR, Keil and GCC.

5.6 How are the product/peripheral specific features managed?

The HAL offers extended APIs, that is, specific functions as add-ons to the common API to support features available on some products/lines only.

5.7 How can STM32CubeMX generate code based on embedded software?

STM32CubeMX has a built-in knowledge of STM32 microcontrollers, including their peripherals and software. This enables the tool to provide a graphical representation to the user and generate *.h/*.c files based on the user configuration.

5.8 How can the user get regular updates on the latest STM32CubeF3 firmware releases?

The STM32CubeF3 firmware package comes with an updater utility, STM32CubeUpdater, that can be configured for automatic or on-demand checks for new firmware package updates (new releases or/and patches).

STM32CubeUpdater is also integrated within the STM32CubeMX tool. When using this tool for STM32F3 configuration and initialization C code generation, the user can benefit from STM32CubeMX self-updates as well as STM32CubeF3 firmware package updates.

For more details, refer to *Section 4.4*.

5.9 How can the user set the HAL drivers in Debug mode to debug his/her application?

The HAL drivers can be configured in Debug mode by setting the USE_FULL_ASSERT compilation switch in the user application project environment. This allows to systematically check the parameters passed to the HAL APIs. The assert_failed() function is called when an error is detected.

Note: Because of the overhead run-time checking introduces, it is recommended to use it during application code development and debugging, and to remove it from the final application to improve code size and speed.

6 Revision history

Table 4. Document revision history

Date	Revision	Changes
17-Jun-2014	1	Initial release.
20-Nov-2014	2	Updated middleware list in *Section : Introduction*. Updated *Figure 2: STM32CubeF3 firmware architecture* Added STM32F302xD, STM32F302xE, STM32F303xD, STM32F303xE and STM32F398xx in *Table 1: Macros for STM32F3 series*. Renamed STM32F3-DISCOVERY and STM32F3348-Discovery part numbers into STM32F3DISCOVERY and 32F3348DISCOVERY, respectively. *Section 3.1: Supported STM32F3 devices and hardware*: updated *Table 2: Boards for STM32F3 series*; replaced NUCLEO-L302R8 and NUCLEO-L334R8 by NUCLEO-F302R8, NUCLEO-F334R8 and NUCLEO-F303RE. Updated *Table 3: Number of examples available for each board*. Updated *Section 3.2: Firmware package overview*. Updated *Section 5: FAQs*.
12-Jun-2015	3	Added SW4STM32 in *Section 3.2: Firmware package overview* and *Section 4.1: Running your first example*. Updated *Figure 4: STM32CubeF3 example overview*.
10-Sep-2015	4	Updated *Figure 1: STM32Cube™ firmware components*. Added NUCLEO-F303K8 board.
09-Nov-2015	5	Added NUCLEO-F303ZE in *Table 2: Boards for STM32F3 series*. Added Nucleo-144 in *Section 3.1: Supported STM32F3 devices and hardware*. Updated *Table 3: Number of examples available for each board*. Updated list of STM32F3 boards in *Section 5.2: Which boards are supported by the STM32CubeF3 firmware package?*.